```
    return
endif
jm=(jhi+jlo)/2
if(x.ge.xx(jm).eqv.ascnd)then
    jlo=jm
else
    jhi=jm
endif
goto 3
END
```

After the Hunt

The problem: Routines `locate` and `hunt` return an index `j` such that your desired value lies between table entries `xx(j)` and `xx(j+1)`, where `xx(1:n)` is the full length of the table. But, to obtain an `m`-point interpolated value using a routine like `polint` (§3.1) or `ratint` (§3.2), you need to supply much shorter `xx` and `yy` arrays, of length `m`. How do you make the connection?

The solution: Calculate

```
k = min(max(j-(m-1)/2,1),n+1-m)
```

This expression produces the index of the leftmost member of an `m`-point set of points centered (insofar as possible) between `j` and `j+1`, but bounded by 1 at the left and `n` at the right. FORTRAN then lets you call the interpolation routine with array addresses offset by `k`, e.g.,

```
call polint(xx(k),yy(k),m,...)
```

CITED REFERENCES AND FURTHER READING:

Knuth, D.E. 1973, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley), §6.2.1.

3.5 Coefficients of the Interpolating Polynomial

Occasionally you may wish to know not the value of the interpolating polynomial that passes through a (small!) number of points, but the coefficients of that polynomial. A valid use of the coefficients might be, for example, to compute simultaneous interpolated values of the function and of several of its derivatives (see §5.3), or to convolve a segment of the tabulated function with some other function, where the moments of that other function (i.e., its convolution with powers of x) are known analytically.

However, please be certain that the coefficients are what you need. Generally the coefficients of the interpolating polynomial can be determined much less accurately than its value at a desired abscissa. Therefore it is not a good idea to determine the coefficients only for use in calculating interpolating values. Values thus calculated will not pass exactly through the tabulated points, for example, while values computed by the routines in §3.1–§3.3 will pass exactly through such points.

Also, you should not mistake the interpolating polynomial (and its coefficients) for its cousin, the *best fit* polynomial through a data set. Fitting is a *smoothing* process, since the number of fitted coefficients is typically much less than the number of data points. Therefore, fitted coefficients can be accurately and stably determined even in the presence of statistical errors in the tabulated values. (See §14.8.) Interpolation, where the number of coefficients and number of tabulated points are equal, takes the tabulated values as perfect. If they in fact contain statistical errors, these can be magnified into oscillations of the interpolating polynomial in between the tabulated points.

As before, we take the tabulated points to be $y_i \equiv y(x_i)$. If the interpolating polynomial is written as

$$y = c_1 + c_2 x + c_3 x^2 + \cdots + c_N x^{N-1} \tag{3.5.1}$$

then the c_i's are required to satisfy the linear equation

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{N-1} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.5.2}$$

This is a *Vandermonde matrix*, as described in §2.8. One could in principle solve equation (3.5.2) by standard techniques for linear equations generally (§2.3); however the special method that was derived in §2.8 is more efficient by a large factor, of order N, so it is much better.

Remember that Vandermonde systems can be quite ill-conditioned. In such a case, *no* numerical method is going to give a very accurate answer. Such cases do not, please note, imply any difficulty in finding interpolated *values* by the methods of §3.1, but only difficulty in finding *coefficients*.

Like the routine in §2.8, the following is due to G.B. Rybicki.

```
SUBROUTINE polcoe(x,y,n,cof)
INTEGER n,NMAX
REAL cof(n),x(n),y(n)
PARAMETER (NMAX=15)           Largest anticipated value of n.
   Given arrays x(1:n) and y(1:n) containing a tabulated function y_i = f(x_i), this routine
   returns an array of coefficients cof(1:n), such that y_i = sum_j cof_j x_i^(j-1).
INTEGER i,j,k
REAL b,ff,phi,s(NMAX)
do 11 i=1,n
    s(i)=0.
    cof(i)=0.
enddo 11
s(n)=-x(1)
do 13 i=2,n                   Coefficients s_i of the master polynomial P(x) are found
    do 12 j=n+1-i,n-1           by recurrence.
        s(j)=s(j)-x(i)*s(j+1)
    enddo 12
    s(n)=s(n)-x(i)
enddo 13
do 16 j=1,n
    phi=n
    do 14 k=n-1,1,-1          The quantity phi = prod_{j≠k}(x_j - x_k) is found as a deriva-
                                tive of P(x_j).
```

```
            phi=k*s(k+1)+x(j)*phi
        enddo 14
        ff=y(j)/phi
        b=1.                                 Coefficients of polynomials in each term of the Lagrange
        do 15 k=n,1,-1                           formula are found by synthetic division of P(x) by
            cof(k)=cof(k)+b*ff                   (x - x_j). The solution c_k is accumulated.
            b=s(k)+x(j)*b
        enddo 15
    enddo 16
    return
    END
```

Another Method

Another technique is to make use of the function value interpolation routine already given (polint §3.1). If we interpolate (or extrapolate) to find the value of the interpolating polynomial at $x = 0$, then this value will evidently be c_1. Now we can subtract c_1 from the y_i's and divide each by its corresponding x_i. Throwing out one point (the one with smallest x_i is a good candidate), we can repeat the procedure to find c_2, and so on.

It is not instantly obvious that this procedure is stable, but we have generally found it to be somewhat *more* stable than the routine immediately preceding. This method is of order N^3, while the preceding one was of order N^2. You will find, however, that neither works very well for large N, because of the intrinsic ill-condition of the Vandermonde problem. In single precision, N up to 8 or 10 is satisfactory; about double this in double precision.

```
      SUBROUTINE polcof(xa,ya,n,cof)
      INTEGER n,NMAX
      REAL cof(n),xa(n),ya(n)
      PARAMETER (NMAX=15)                    Largest anticipated value of n.
C     USES polint
```

Given arrays xa(1:n) and ya(1:n) of length n containing a tabulated function $ya_i = f(xa_i)$, this routine returns an array of coefficients cof(1:n), also of length n, such that $ya_i = \sum_j cof_j xa_i^{j-1}$.

```
      INTEGER i,j,k
      REAL dy,xmin,x(NMAX),y(NMAX)
      do 11 j=1,n
          x(j)=xa(j)
          y(j)=ya(j)
      enddo 11
      do 14 j=1,n
          call polint(x,y,n+1-j,0.,cof(j),dy)    This is the polynomial interpolation rou-
          xmin=1.e38                                 tine of §3.1. We extrapolate to x =
          k=0                                        0.
          do 12 i=1,n+1-j                        Find the remaining x_i of smallest abso-
              if (abs(x(i)).lt.xmin)then             lute value,
                  xmin=abs(x(i))
                  k=i
              endif
              if(x(i).ne.0.)y(i)=(y(i)-cof(j))/x(i)    (meanwhile reducing all the terms)
          enddo 12
          do 13 i=k+1,n+1-j                          and eliminate it.
              y(i-1)=y(i)
              x(i-1)=x(i)
          enddo 13
      enddo 14
      return
      END
```

If the point $x = 0$ is not in (or at least close to) the range of the tabulated x_i's, then the coefficients of the interpolating polynomial will in general become very large. However, the real "information content" of the coefficients is in small differences from the "translation-induced" large values. This is one cause of ill-conditioning, resulting in loss of significance and poorly determined coefficients. You should consider redefining the origin of the problem, to put $x = 0$ in a sensible place.

Another pathology is that, if too high a degree of interpolation is attempted on a smooth function, the interpolating polynomial will attempt to use its high-degree coefficients, in combinations with large and almost precisely canceling combinations, to match the tabulated values down to the last possible epsilon of accuracy. This effect is the same as the intrinsic tendency of the interpolating polynomial values to oscillate (wildly) between its constrained points, and would be present even if the machine's floating precision were infinitely good. The above routines `polcoe` and `polcof` have slightly different sensitivities to the pathologies that can occur.

Are you still quite certain that using the *coefficients* is a good idea?

CITED REFERENCES AND FURTHER READING:

Isaacson, E., and Keller, H.B. 1966, *Analysis of Numerical Methods* (New York: Wiley), §5.2.

3.6 Interpolation in Two or More Dimensions

In multidimensional interpolation, we seek an estimate of $y(x_1, x_2, \ldots, x_n)$ from an n-dimensional grid of tabulated values y and n one-dimensional vectors giving the tabulated values of each of the independent variables $x_1, x_2, \ldots, x_n$. We will not here consider the problem of interpolating on a mesh that is not Cartesian, i.e., has tabulated function values at "random" points in n-dimensional space rather than at the vertices of a rectangular array. For clarity, we will consider explicitly only the case of two dimensions, the cases of three or more dimensions being analogous in every way.

In two dimensions, we imagine that we are given a matrix of functional values `ya(j,k)`, where `j` varies from 1 to `m`, and `k` varies from 1 to `n`. We are also given an array `x1a` of length `m`, and an array `x2a` of length `n`. The relation of these input quantities to an underlying function $y(x_1, x_2)$ is

$$\texttt{ya(j,k)} = y(\texttt{x1a(j)}, \texttt{x2a(k)}) \tag{3.6.1}$$

We want to estimate, by interpolation, the function y at some untabulated point (x_1, x_2).

An important concept is that of the *grid square* in which the point (x_1, x_2) falls, that is, the four tabulated points that surround the desired interior point. For convenience, we will number these points from 1 to 4, counterclockwise starting from the lower left (see Figure 3.6.1). More precisely, if

$$\begin{aligned} \texttt{x1a(j)} \le x_1 \le \texttt{x1a(j+1)} \\ \texttt{x2a(k)} \le x_2 \le \texttt{x2a(k+1)} \end{aligned} \tag{3.6.2}$$